Steven M. Hoffberg

From: Steven M. Hoffberg [steve@hoffberg.org]
Sent: Thursday, November 18, 2004 12:18 PM
To: 'Nguyen, Nga'
Subject: 09/599,163 tvs_client.c

```
/* ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 * tvs_client - implements the client interface to TVS
 * Copyright (c) 1995 Newshare Corporation
 * ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <bstring.h>
#include <time.h>
#include <ctype.h>
#include <signal.h>
#include <sys/un.h>
#include <sys/errno.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netdb.h>
#ifdef SOLARIS2
#include <arpa/inet.h>
#endif /* SOLARIS2 */
#ifdef IRIX
#include <unistd.h>
#endif /* IRIX */

#include "tvs.h"
#include "tvs_client.h"
#include "tvs_config.h"
#include "tvs_log.h"
#include "tvs_error.h"
#include "tvs_profile.h"
#include "tvs_util.h"

#define TVSC_MAX_TOKEN_LENGTH 72
#define TVSC_MAX_BUFLEN TVSC_MAX_TOKEN_LENGTH*2
```

```
#define TVS_MAX_CONNECT_RETRIES 4

#define FREE(x)   free(x)            /* so I can use my own for debug */
#define MALLOC(x) malloc(x)

PRIVATE int tvs_is_initialized = 0;
PRIVATE int service_initialized = 0;

/* address structures for myself, and my TVS server (once obtained) */

PRIVATE struct sockaddr_in client_sa, server_sa;
PRIVATE int client_sock, clilen, serlen;

/*
 * server identification information
 * (I get this from tvs_request_service(), and also put it into a disk file)
 */

PRIVATE unsigned short tvs_server_id = 0;
PRIVATE char          *tvs_server_name = (char *) NULL;
PUBLIC TVS_SERVER      tvss_sid = (TVS_SERVER) NULL;

/* everybody's message buffer for stuff returned from TVS */

PRIVATE char response[TVSC_MAX_BUFLEN];
PRIVATE int  resplen = TVSC_MAX_BUFLEN;

#if 0
/* ------------------------------------------------------------------------
 * wrappers
 * ------------------------------------------------------------------------
 */

PRIVATE void
tvs_free(void *mem)
{
  free(mem);
}

PRIVATE void *
tvs_malloc(int mem)
{
  return (void *) malloc(mem);
}
#endif

/* ------------------------------------------------------------------------
 * close up shop
 * ------------------------------------------------------------------------
 */
```

```
PRIVATE int
teardown_service()
{
   close(client_sock);
   client_sock = 0;
   service_initialized = 0;
   return 1;
}

/* ------------------------------------------------------------------------
 * initialize some globally useful structures
 * ------------------------------------------------------------------------
 */

PRIVATE int
init_service()
{
   struct servent *sp;

   bzero(&client_sa, sizeof(struct sockaddr_in));

   /* build and bind my client side */

   client_sa.sin_family = AF_INET;
   client_sa.sin_port   = htons(0);
   client_sa.sin_addr.s_addr = htonl(INADDR_ANY);

   clilen = sizeof(struct sockaddr_in);

   client_sock = socket(AF_INET, SOCK_DGRAM, 0);
   if (!client_sock) return 0;

   if (bind(client_sock, (struct sockaddr *) &client_sa, clilen) < 0)
    return 0;

   /* set up server boilerplate */

   bzero(&server_sa, sizeof(struct sockaddr_in));
   serlen = sizeof(struct sockaddr_in);

   /*
   sp = getservbyname(SERVICE_NAME, "udp");
   if (!sp) return 0;
   */

   sp = (struct servent *) MALLOC(sizeof(struct servent));

   sp->s_port = htons(CLICKSHARE_PORT);

   server_sa.sin_port = sp->s_port;
   server_sa.sin_family = AF_INET;
```

```
  FREE(sp);

  service_initialized |= 1;
  return 1;
}

/* ---------------------------------------------------------------------------
 * handle this boring name/addr lookup stuff.
 * ---------------------------------------------------------------------------
 */

PRIVATE int
build_server_addr(char *server)
{
  struct hostent *hp;

  hp = gethostbyname(server);
  if (hp == (struct hostent *) NULL) {
    sprintf(msgString, "tvsc_request_service: can find server host [%s]\n",
            server);
    LogMsg(LOG_INFO, msgString);
    return 0;
  }

  bcopy(hp->h_addr_list[0], &(server_sa.sin_addr.s_addr), hp->h_length);
#if DEBUG
  printf("using server addr: %s\n", inet_ntoa(server_sa.sin_addr));
#endif
  return 1;
}

/* ---------------------------------------------------------------------------
 * response packets from TVS contain an ACK as the first character of the
 * packet. This ACK is the encode_command() form of enum request_type if
 * the command was successful, OR the same logically ORed with TVS_NAK if not.
 * If not successful, the remainder of the packet is an error code in
 * character form (as found in tvs_error.h).
 * ---------------------------------------------------------------------------
 */

PRIVATE int
tvsc_success(char cmd, char response)
{
  /* response contains NAK */

#ifdef DEBUG
  fprintf(stderr,"cmd: %c response: %c (0x%x)\n", cmd, response,
          (response & TVS_NAK));
#endif /* DEBUG */
  if ((cmd & TVS_NAK)) return 0;
```

```
  /* make sure that the response actually contains the command */

  if (cmd != response) return 0;

  return 1;
}

/* ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 * here's the main functionality
 * ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 */

/* ---------------------------------------------------------------------------
 * PACKET FORMAT
 * cmd 1 char
 * host IP 4 bytes
 * PM id 4 chars
 * ---------------------------------------------------------------------------
 */

PRIVATE int
tvsc_request_service()
{
  extern int tvs_server_error;

  char cmd;
  char **servers;          /* will be NULL terminated list of hostnames */
  struct iovec iobuf[3];
  unsigned long haddr;

  int got_response = 0;
  unsigned long id_pm;

  /* set up net, and a list of servers to try */

  if (!service_initialized)
   (void) init_service();

  servers = tvs_servers;
  if (!servers) return 0;

  tvs_server_error = TVS_NO_SERVER_ERROR;

  /* create my packet */

  cmd = encode_command(TVS_REQUEST_SERVICE);
  iobuf[0].iov_base = (caddr_t) &cmd;              /* command */
  iobuf[0].iov_len  = 1;
```

```
  /* my own address */

  if (!get_my_address(&haddr)) {
    LogMsg(LOG_ERR, "server unable to get own host address.");
    return 0;
  }

  iobuf[1].iov_base = (caddr_t) &haddr;
  iobuf[1].iov_len  = sizeof(unsigned long);

  /* my PM id */

  id_pm = htonl(tvs_pm_id);
  iobuf[2].iov_base = (caddr_t) &id_pm;
  iobuf[2].iov_len  = sizeof(unsigned long);

  /* try each of our servers till we get a token or die */

  while(*servers) {
    if(!build_server_addr(*servers)) {
        servers++;
        continue;
    }
#ifdef DEBUG
    fprintf(stderr,"request service: trying: %s\n", *servers);
#endif /* DEBUG */
    if (connect(client_sock, (struct sockaddr *) &server_sa, serlen) < 0) {
      perror("connect");
        servers++;
        continue;
    }

    resplen = TVSC_MAX_BUFLEN;
    if (rdp_vtransact(client_sock, &iobuf[0], 3, response, &resplen) > 0) {
#ifdef DEBUG
        fprintf(stderr,"got response: %c: ", response[0]);
        if (isalpha(response[1]))
         fprintf(stderr,"%s\n", &response[1]);
        else
         fprintf(stderr,"0x%x 0x%x 0x%x 0x%x 0x%x 0x%x\n",
                response[1],response[2],response[3], response[4],
                response[5],response[6]);
#endif /* DEBUG */
        response[resplen] = '\0';
        got_response++;
        break;
    }
    else
     servers++;
  }
```

```
  if (!got_response)
    return 0;

  if (!tvsc_success(cmd, response[0])) {
    tvs_set_return_string(&response[1]);
    return 0;
  }

  /*
   * store up information about the server we've connected to
   * NOTE: I will need this later if I have to restart
   * NOTE: address stays in net byte order, id converts to host.
   */

  memcpy(&tvs_server_id, &response[5], sizeof(unsigned short));

  tvs_server_name = (char *) MALLOC(strlen(*servers)+1);
  strcpy(tvs_server_name, *servers);
  tvss_sid = tvs_new_server_id(server_sa.sin_addr.s_addr,
                               ntohs(tvs_server_id));

  /* save this connection name for next time */
  tvs_save_server_name(tvss_sid);

  sprintf(msgString, "obtained %s for Clickshare service", tvs_server_name);
  LogMsg(LOG_NOTICE, msgString);
  return 1;
}

/* ---------------------------------------------------------------------------
 * PACKET FORMAT
 * cmd 1 char
 * host IP 4 bytes
 * PM id 4 chars
 * ---------------------------------------------------------------------------
 */

PRIVATE int
tvsc_drop_service()
{
  char cmd;
  struct iovec iobuf[3];
  unsigned long haddr;
  unsigned long id_pm;

  cmd = encode_command(TVS_DROP_SERVICE);
  iobuf[0].iov_base = (caddr_t) &cmd;                    /* command */
  iobuf[0].iov_len  = 1;

  if (!get_my_address(&haddr)) {
```

```
    LogMsg(LOG_ERR, "server unable to get own host address.");
    exit(1);
  }

  iobuf[1].iov_base = (caddr_t) &haddr;
  iobuf[1].iov_len  = sizeof(unsigned long);

  id_pm = htonl(tvs_pm_id);
  iobuf[2].iov_base = (caddr_t) &id_pm;
  iobuf[2].iov_len  = sizeof(unsigned long);

  resplen = TVSC_MAX_BUFLEN;
  rdp_vtransact(client_sock, &iobuf[0], 3, response, &resplen);
  response[resplen] = '\0';
  tvs_set_return_string(&response[1]);
  if (!tvsc_success(cmd, response[0])) {
    sprintf(msgString,"drop error: %s (%d)\n", &response[1], resplen);

    /* it is not always fatal if the drop_service() fails, so just notify */

    LogMsg(LOG_NOTICE, msgString);
    return 0;
  }

  /* clear out current service IDs */

  tvs_server_id = 0;
  tvs_server_name = (char *) NULL;

  /* release everything */

  teardown_service();                    /* so I can "clean start" */
  tvs_unlink_server_name_file();
  return 1;
}


/* ---------------------------------------------------------------------------
 * This is called first (upon startup) to assure that we start a clean
 * session with TVS.  This takes care of problems when/if previous server
 * invocations die abnormally.
 * ---------------------------------------------------------------------------
 */

PRIVATE int
tvsc_invalidate_service()
{
  char *server_host;
  int tvss_id;
  unsigned long ss_addr;
```

```
  server_host = tvs_load_server_name (&tvss_id);
  if (!server_host)
    return 1;

  /* if I have successfully read contents, assume I had better contact
   * the old server to drop my old "connection".
   */

  init_service();

  if (!strncmp(server_host, "0x", 2)) {
     sscanf(server_host, "0x%lx", &ss_addr);
     bcopy((char *) &ss_addr, &(server_sa.sin_addr.s_addr),
           sizeof(unsigned long));
  }
  else
    build_server_addr(server_host);

  if (connect(client_sock, (struct sockaddr *) &server_sa, serlen) < 0)
   return -1;

  /* let drop service handle all this */

  tvsc_drop_service();
  return 1;
}

/* ---------------------------------------------------------------------------
 * PACKET FORMAT
 * cmd 1 char
 * user's host IP 4 bytes  (in the profile)
 * user's profile (encoded) variable bytes
 * ---------------------------------------------------------------------------
 */

PRIVATE TVS_TOKEN
tvsc_request_token(TVS_PROFILE prof)
{
  char *request;
  char cmd;
  struct iovec iobuf[2];
  TVS_TOKEN token;

  if (!tvs_is_initialized)
    return 0;

  cmd = encode_command(TVS_NEW_TOKEN);
  iobuf[0].iov_base = (caddr_t) &cmd;                    /* command */
  iobuf[0].iov_len  = 1;

  request = tvs_encode_profile(prof); /* encode the profile we're sending */
```

```
  if (!request) return 0;

  iobuf[1].iov_base = (caddr_t) request;
  iobuf[1].iov_len  = strlen(request);

  resplen = TVSC_MAX_BUFLEN;
  rdp_vtransact(client_sock, &iobuf[0], 2, response, &resplen);
  response[resplen] = '\0';
  tvs_set_return_string(&response[1]);
  if (tvsc_success(cmd, response[0])) {
    token = (char *) MALLOC(strlen(response));
    strcpy(token, &response[1]);
  }
  else {
    sprintf(msgString, "request token error: %s (%d)\n",
            &response[1], resplen);
    LogMsg(LOG_ERR, msgString);
    token = (TVS_TOKEN) NULL;
  }

  FREE(request);
  return token;
}

/* ---------------------------------------------------------------------------
 * PACKET FORMAT
 * cmd 1 char
 * user's host ip addr (incoming connection) 4 bytes
 * user's token (variable bytes)
 * ---------------------------------------------------------------------------
 */

PRIVATE TVS_PROFILE
tvsc_request_validation(TVS_TOKEN token, unsigned long host_id)
{
  char cmd;
  struct iovec iobuf[3];
  TVS_PROFILE prof;

  if (!tvs_is_initialized)
    return 0;

  cmd = encode_command(TVS_VALIDATE_TOKEN);
  iobuf[0].iov_base = (caddr_t) &cmd;                    /* command */
  iobuf[0].iov_len  = 1;

  iobuf[1].iov_base = (caddr_t) &host_id;
  iobuf[1].iov_len  = sizeof(unsigned long);

  iobuf[2].iov_base = (caddr_t) token;
  iobuf[2].iov_len  = strlen(token);
```

```
  resplen = TVSC_MAX_BUFLEN;
  rdp_vtransact(client_sock, &iobuf[0], 3, response, &resplen);
  response[resplen] = '\0';
  tvs_set_return_string(&response[1]);
  if (tvsc_success(cmd, response[0])) {
    prof = tvs_decode_profile(&response[1]);
  }
  else {
    /* server return message will probably contain address to forward to */
    sprintf(msgString,"error validating token: %s\n",
            tvs_get_server_return());
    LogMsg(LOG_NOTICE, msgString);

    prof = (TVS_PROFILE) NULL;
  }

  return prof;
}

/* ---------------------------------------------------------------------------
 * PACKET FORMAT
 * cmd 1 char
 * user's token (variable bytes)
 * reason 4 bytes
 * ---------------------------------------------------------------------------
 */

PRIVATE int
tvsc_request_invalidate(TVS_TOKEN token, int reason)
{
  char cmd;
  struct iovec iobuf[2];

  if (!tvs_is_initialized)
    return 0;

  cmd = encode_command(TVS_INVALIDATE_TOKEN);
  iobuf[0].iov_base = (caddr_t) &cmd;                    /* command */
  iobuf[0].iov_len  = 1;

  iobuf[1].iov_base = (caddr_t) token;
  iobuf[1].iov_len  = strlen(token);

  /* reason not currently used */

/* iobuf[2].iov_base = (caddr_t) &reason;
 * iobuf[2].iov_len  = sizeof(int);
 */

  resplen = TVSC_MAX_BUFLEN;
```

```
  rdp_vtransact(client_sock, &iobuf[0], 2, response, &resplen);
  response[resplen] = '\0';
  tvs_set_return_string(&response[1]);
  if (tvsc_success(cmd, response[0])) {
    return 1;
  }
  else {
   sprintf(msgString,"error invalidating token: %s (%d)\n",
            &response[1], resplen);
   LogMsg(LOG_NOTICE, msgString);
  }

  return 0;
}

/* ---------------------------------------------------------------------------
 * PACKET FORMAT
 * cmd 1 char
 * server id (variable bytes)
 * ---------------------------------------------------------------------------
 */

PRIVATE char *
tvsc_request_id(TVS_SERVER tvs_id)
{
  char cmd;
  struct iovec iobuf[2];
  unsigned short nid;
  char *resp = (char *) NULL;

  if (!tvs_is_initialized)
    return 0;

  cmd = encode_command(TVS_IDENTIFY_SERVICE);
  iobuf[0].iov_base = (caddr_t) &cmd;
  iobuf[0].iov_len = 1;

  nid = tvs_get_server_id(tvs_id);
  nid = htons(nid);
  iobuf[1].iov_base = (caddr_t) &nid;
  iobuf[1].iov_len = sizeof(unsigned short);

  resplen = TVSC_MAX_BUFLEN;
  rdp_vtransact(client_sock, &iobuf[0], 2, response, &resplen);
  response[resplen] = '\0';
  tvs_set_return_string(&response[1]);
  if (tvsc_success(cmd, response[0])) {
    resp = (char *) MALLOC(resplen);
    strcpy(resp, &response[1]);
  }
  else {
```

```
    sprintf(msgString, "error getting TVS server ID: %s (%d)\n",
            &response[1], resplen);
    LogMsg(LOG_NOTICE, msgString);
  }

  return resp;
}

/* ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 * Public interface to the TVS server
 * (These are really just error wrappers around the "raw" client functions).
 * ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 */

/* ---------------------------------------------------------------------------
 * tvs_initialize_service - start a session with TVS
 * ---------------------------------------------------------------------------
 */

PUBLIC TVS_SERVER
tvs_initialize_service(char *tvs_conf)
{
  extern int tvs_error;

  tvs_error = TVS_NO_ERROR;

  /* locate all my configuration files */

  if (!tvs_read_config(tvs_conf)) {
    LogMsg(LOG_ERR, "problem reading server configuration file:");
    LogMsg(LOG_ERR, tvs_error_msg);
    return (TVS_SERVER) NULL;
  }

  /* try to patch things up if a previous session with TVS crashed */

  tvsc_invalidate_service();

  /* now, request a "fresh service" */

  if (tvsc_request_service())
    tvs_is_initialized |= 1;
  else {
    tvs_error = TVS_SERVER_ERROR;
    tvs_is_initialized = 0;
  }

  return tvss_sid;
}
```

```
/* ---------------------------------------------------------------------------
 * tvs_new_token - ask the TVS server for a new TVS token for a given user.
 * ---------------------------------------------------------------------------
 */

PUBLIC TVS_TOKEN
tvs_new_token(TVS_PROFILE prof)
{
   extern int tvs_error;
   TVS_TOKEN token;

   tvs_error = TVS_NO_ERROR;

   if (!prof) {
      tvs_error = TVS_PROFILE_NOT_PROVIDED;
      return (TVS_TOKEN) NULL;
   }

   if (!tvs_is_initialized) {
      tvs_error = TVS_SERVER_NOT_INITIALIZED;
      return (TVS_TOKEN) NULL;
   }

   if ((token = tvsc_request_token(prof)) == (TVS_TOKEN) NULL) {
      tvs_error = TVS_SERVER_ERROR;
      return (TVS_TOKEN) NULL;
   }

   return token;
}

/* ---------------------------------------------------------------------------
 * tvs_validate_token - ask TVS if a given token is valid
 * ---------------------------------------------------------------------------
 */

PUBLIC TVS_PROFILE
tvs_validate_token(TVS_TOKEN token, unsigned long h_id)
{
   extern int tvs_error;
   TVS_PROFILE prof;

   tvs_error = TVS_NO_ERROR;

   if (!token) {
      tvs_error = TVS_INVALID_TOKEN_PROVIDED;
      return (TVS_PROFILE) NULL;
   }

   if (!tvs_is_initialized) {
```

```
    tvs_error = TVS_SERVER_NOT_INITIALIZED;
    return (TVS_PROFILE) NULL;
  }

  if ((prof = tvsc_request_validation(token, h_id)) == (TVS_PROFILE) NULL) {
    tvs_error = TVS_SERVER_ERROR;
    return (TVS_PROFILE) NULL;
  }

  return prof;
}

/* ---------------------------------------------------------------------------
 * tvs_invalidate_token - ask TVS to trash a given token
 * ---------------------------------------------------------------------------
 */

PUBLIC int
tvs_invalidate_token(TVS_TOKEN token, int reason)
{
  extern int tvs_error;
  tvs_error = TVS_NO_ERROR;

  if (!token) {
    tvs_error = TVS_INVALID_TOKEN_PROVIDED;
    return 0;
  }

  if (!tvs_is_initialized) {
    tvs_error = TVS_SERVER_NOT_INITIALIZED;
    return 0;
  }

  if (!tvsc_request_invalidate(token, reason)) {
    tvs_error = TVS_SERVER_ERROR;
    return 0;
  }
  return 1;
}

/* ---------------------------------------------------------------------------
 * tvs_identify_tvs_server - get the id string of a given TVS server
 * ---------------------------------------------------------------------------
 */

PUBLIC char *
tvs_identify_tvs_server()
{
  char *id;
  extern int tvs_error;
```

```
  tvs_error = TVS_NO_ERROR;

  if (!tvss_sid) {
    tvs_error = TVS_NO_SERVER_ID_GIVEN;
    return (char *) NULL;
  }

  if (!tvs_is_initialized) {
    tvs_error = TVS_SERVER_NOT_INITIALIZED;
    return  (char *) NULL;
  }

  id = tvsc_request_id(tvss_sid);
  if (!id)
   tvs_error = TVS_SERVER_ERROR;

  return id;
}

/* ---------------------------------------------------------------------------
 * tvs_drop_service - cleanly disconnect from the TVS server
 * ---------------------------------------------------------------------------
 */

PUBLIC void
tvs_drop_service()
{
  extern int tvs_error;

  tvs_error = TVS_NO_ERROR;
  tvsc_drop_service();
  return;
}

/* ---------------------------------------------------------------------------
 * tvs_invalidate_service - clean-up after a bad disconnect from the TVS
 * server (OK, to call this under ANY circumstance at startup -- before
 * tvs_initialize_server).
 * ---------------------------------------------------------------------------
 */

PUBLIC void
tvs_invalidate_service()
{
  extern int tvs_error;
  tvs_error = TVS_NO_ERROR;

  tvsc_invalidate_service();
  return;
}
```

```
/* ---------------------------------------------------------------------------
 * tvs_destroy_token - get rid of this token
 * ---------------------------------------------------------------------------
 */

PUBLIC void
tvs_destroy_token(TVS_TOKEN token)
{
   FREE(token);
}
```

Very truly yours,

Steven M. Hoffberg
Milde & Hoffberg, LLP
Suite 460
10 Bank Street
White Plains, NY 10606
(914) 949-3100 tel.
(914) 949-3416 fax
steve@hoffberg.org
www.hoffberg.org